

Durée : 3 heures

 Documents non autorisés

Il sera tenu compte dans la notation de la clarté et de la qualité de la présentation.

Exercice 1

Dans une grammaire attribuée, expliquer ce que sont des attributs synthésés et des attributs hérités. Montrer comment ces attributs peuvent être utilisés dans le contexte de l'analyse syntaxique ascendante.

Exercice 2

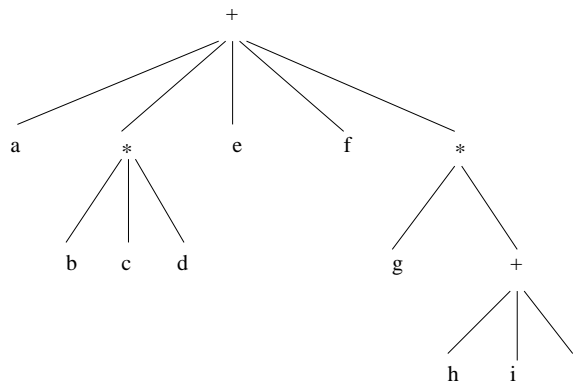
Considérer la grammaire suivante définissant des nombres binaires à virgule :

$$\begin{aligned}
 S &\longrightarrow A, A \mid A \\
 A &\longrightarrow A B \mid B \\
 B &\longrightarrow 0 \mid 1
 \end{aligned}$$

Définir un schéma de traduction en analyse ascendante qui permette de calculer la valeur décimale d'un nombre binaire à virgule. Ainsi le nombre binaire 101,101 a pour valeur décimale 5,625. Montrer comment fonctionne ce schéma sur le nombre binaire 1100,01.

Exercice 3

Dans cet exercice, on considère des expressions arithmétiques quelconques, formées des opérateurs + et *, sur lesquelles s'appliquent les règles standards de priorité et d'associativité. L'objectif de cet exercice est de déterminer pour toute expression arithmétique, son *arbre syntaxique abstrait réduit* dans lequel toutes les additions ou multiplications de même niveau sont représentées par un seul noeud. Ainsi l'expression $a + b * c * d + e + f + g * (h + i + j)$ est représentée par l'arbre réduit ci-dessous.



A partir de la grammaire non ambiguë des expressions arithmétiques, définir un schéma de traduction en analyse ascendante permettant de construire de tels arbres syntaxique abstraits réduits. Préciser la structure de données utilisée pour stocker cet arbre, ainsi que les opérations permettant de le manipuler.

Exercice 4

On considère un langage dans lequel un identificateur peut désigner soit une variable simple, soit une matrice, dans tous les cas de type entier. On fait l'hypothèse que les informations concernant ces variables ont été stockées dans la table des symboles (lors de l'analyse de leur déclaration). Pour les matrices, sont en particulier stockées leurs dimensions.

L'instruction $C = A + B$ désigne alors soit l'addition de deux entiers, soit une addition matricielle, selon la nature des variables A, B et C .

On fait les hypothèses suivantes. Les matrices sont stockées en mémoire selon le mode dit "du balayage TV" où les éléments sont stockés par lignes d'abord. Un entier est stocké sur 4 octets et la mémoire est adressable par octets.

(1) A partir de la grammaire simplifiée suivante, définir un schéma de traduction en analyse ascendante, qui génère le code intermédiaire sous forme de quadruplets (selon le même principe que celui présenté en cours),

$$S \longrightarrow id = id + id \mid id = id * id$$

en précisant clairement quels sont les fonctions et les attributs utilisés.

(2) Expliquer comment le code produit peut être optimisé.

Exercice 5

Dans le schéma de traduction proposé en cours pour les expressions booléennes, le code intermédiaire produit n'est pas toujours optimal, en particulier lorsque l'on analyse les `goto` produits. Ainsi l'expression booléenne $((A=B) \text{ or } (C=D)) \text{ and not } (E=F) \text{ or } ((G=H) \text{ and } (I=J))$ est traduite par :

```
100 : if A = B goto 104
101 : goto 102
102 : if C = D goto 104
103 : goto 106
104 : if E = F goto 106
105 : goto TRUE
106 : if G = H goto 108
107 : goto FALSE
108 : if I = J goto TRUE
109 : goto FALSE
```

Une optimisation simple permettrait de supprimer les quadruplets de la forme $L : \text{goto } L+1$, soit dans cet exemple le quadruplet d'étiquette 101. Cependant, une solution plus efficace consiste à produire directement le code intermédiaire optimisé suivant, sans `goto inutiles` :

```
100 : if A = B goto 102
101 : if C <> D goto 103
102 : if E <> F goto TRUE
103 : if G <> H goto FALSE
104 : if I <> J goto FALSE
TRUE :
OU
100 : if A = B goto 102
101 : if C <> D goto 103
102 : if E <> F goto TRUE
103 : if G <> H goto FALSE
104 : if I = J goto TRUE
FALSE :
```

où `TRUE` et `FALSE` désignent l'étiquette de l'instruction élémentaire à exécuter lorsque l'expression booléenne est évaluée à `vrai`, respectivement à `false`.

A partir de la grammaire non ambiguë des expressions booléennes, construire un schéma de traduction qui engendre directement un tel code. Montrer comment il fonctionne sur l'exemple suivant : $((A=B) \text{ or } (C=D)) \text{ and not } (E=F)$.